# THE JPSD EXPERIMENT FEDERATION COMMON SOFTWARE

**Richard A. Briggs**
**Virtual Technology Corporation (VTC)**
**8996 Fern Park Drive**
**Burke, VA 22015**
**rbriggs@virtc.com**

**Gordon J. Miller**
**Science Applications International Corporation (SAIC)**
**1100 N. Glebe Road, Suite 1100**
**Arlington, VA 22203**
**gmiller@mnsinc.com**

KEYWORDS
FCS, HLA, DIS, RTI, Simulation, Framework, Architecture, CORBA

ABSTRACT
The Federation Common Software (FCS) is an Object-Oriented Framework that provides the common functionality required to integrate simulations with the HLA Run-Time Infrastructure (RTI). FCS encapsulates the RTI interfaces in C++, provides Federation Object Model (FOM) management, supports translation between Simulation and Federation data representation, and performs translation between compile-time and run-time typing of FOM data. The encapsulation of the RTI interfaces automates the initialization of the federation/federate and performs the CORBA object instantiation that is inherent in the current RTI implementation. A FOM manager is implemented to process the FOM/SOM and allow the simulation to query for RTI specific information which is determined at runtime. The FOM management services are coupled with an extensible mechanism for translation between the RTI data types and the simulation data types. The simulation can register specialized translation objects with the framework to convert between simulation and RTI/FOM data representations. The translation mechanism allows a simulation to communicate using its internal naming and unit conventions without modification to comply with a particular instance of a FOM specification. The FCS is currently being used in the JPSD CLCGF Experiment, the HLA IEC Testbed, and the STOW Simulation Support Framework.

## 1.0 INTRODUCTION

As part of the Defense Modeling & Simulation Office (DMSO) prototyping of the High Level Architecture (HLA), the Joint Precision Strike Demonstration (JPSD) program was tasked with integrating the Corps Level Computer-Generated Forces (CLCGF) system with the HLA Run-Time Infrastructure (RTI).

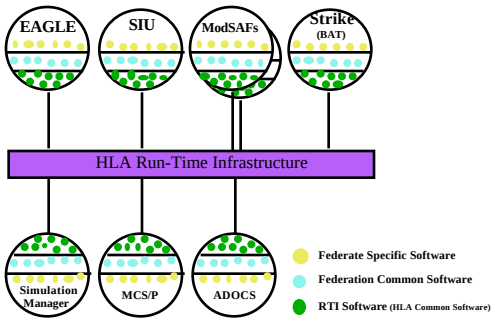


**Figure 1: JPSD Experiment Federation**

CLCGF is a hybrid DIS system, composed of live and aggregate and entity level constructive simulation systems and thus provides a wide variety of interactions that cross many boundaries in distributed simulation (see figure 1).

This paper presents the approach taken by the JPSD team as a case study for future HLA simulation developers and integrators. This paper briefly discusses the components and concepts of HLA, especially where pertinent to additional functionality required of simulations.

## 2.0 HLA RTI OVERVIEW

HLA defines a reusable infrastructure for Modeling and Simulation (M&S) applicable to a wide range of functional applications which facilitate the interoperability and reusability of simulations and systems. The HLA consists of the Object Model Template (OMT), HLA Interface Specification, and HLA Rules. The OMT and HLA Interface Specification will be described with regard to their function and services provided to federates.

### 2.1 Terminology

A brief discussion of common HLA terminology is required to facilitate newcomers' understanding. A group of simulations (know as a system in DIS) is called a *federation* in HLA. Federations are composed of a group of *federates* (simulations and applications in DIS). The data communicated between federates in a federation is documented in a *Federation Object Model* (*FOM*). *Federation execution* is synonymous with a DIS exercise.

### 2.2 Object Model Template

The Object Model Template (OMT) specifies the content and format for key simulation and federation information that is required of all HLA compliant federates and federations. The OMT consists mainly of a set of tables that describe federate and federation data. The OMT tables describe the class inheritance hierarchy of object and interaction class data, the attributes which are members of each object class and the parameters of each interaction class, as well as the interactions between federates. Federation Object Models (FOMs) and Simulation Object Models (SOMs) serve as documentation to facilitate the reuse of simulations.

### 2.3 HLA Interface Specification

The HLA Interface Specification (IF/Spec) describes the set of services supported by an HLA Run-Time Infrastructure (RTI) [1]. There are five service (management) areas in the HLA IF/Spec: Federation Management, Object Management, Declaration Management, Ownership Management, and Time Management.

The Application Programmer Interface (API) is specified (in the HLA IF/Spec) using the Common Object Request Broker Architecture (CORBA) Interface Definition Language (IDL) [4]. IDL is programming language independent and currently has bindings defined for C, C++, and Smalltalk.

Most of the information specified in a SOM and a FOM is not utilized by the RTI. The RTI requires RTI Initialization Data (RID) that consists mainly of the class hierarchies, member attributes/parameters, and default transportation and event ordering information. The RTI has been purposely designed to be a Run-Time Type Identification (RTTI) system so that it requires little Federation data representation information.

## 3.0 FEDERATION COMMON SOFTWARE

The Federation Common Software (FCS) is an extensible Object-Oriented Framework that provides the common functionality required to integrate simulations with the RTI. The FCS encapsulates the RTI interfaces in C++, provides FOM management, supports the translation between Simulation and Federation data representation, and performs translation between compile-time and run-time typing of FOM data.

Figure 2 depicts a layered diagram of the major FCS components. The main FCS functional components deal with the establishment of a communication connection to a federation execution, initialization of the simulation and RTI data type name space, and the translation of data representation throughout the execution of a simulation.

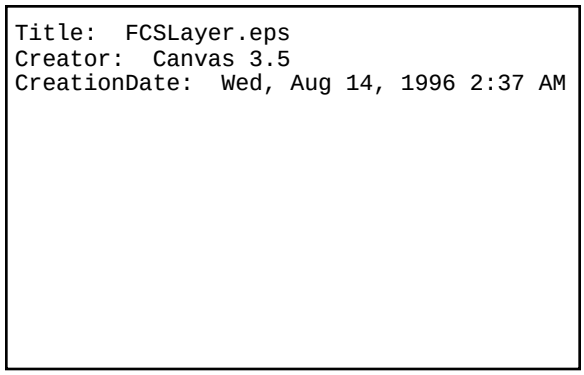


**Figure 2: FCS Layer Diagram**

### 3.1 RTI Encapsulation

The RTI interface was encapsulated to abstract the simulation developer from the current implementation of the RTI interfaces[1]. The v0.X RTI implementation consists of four different components. These components are the Federation Execution, the RTI Executive, the RTI Ambassador, and the Simulation Ambassador. Each of these components must be instantiated in the proper order with the proper arguments. When the Federate object is created it automatically instantiates the RTI components. This can be overridden if the application developer wishes to perform this operation manually. Each of the HLA services are implemented in one of the four RTI objects. The Federate object behaves as a single Facade[2] [5] to each the RTI object interfaces. The Facade allows the Federate object to be the application developers main interface to the FCS and RTI components.

### 3.2 FOM Management and RTI Run-Time Type Identification Services

FOM management is the area in which the FCS provides the most support for integrating or developing a simulation within the HLA. The FCS extends the HLA services and API by providing support for the RTI Run-Time Type Identification and for accessing information contained within the RTI Initialization Data (RID).

#### 3.2.1 Run-Time Type Identification

When a federation execution is created (*Create Federation* service[1]) the RTI assigns a handle (unsigned short) to each of the object and interaction class names and their attribute and parameter names. When the RTI and federates communicate data they use the assigned handles to refer to the type of the object/interaction class or attribute/parameter data. The IF/Spec API section defines methods for getting the handle for a named data type as well as getting the name for a RTI generated handle. Since most applications and programming languages use compile time type identification for data types, several issues arise. First, a mapping must be established to quickly and efficiently translate from RTI to simulation types for data received and to translate from simulation to RTI types for data sent. It is important for this mapping to be optimized since in HLA the type of every attribute of a class is also identified using RTTI. Additional middleware is required to create this mapping and provide it as a service to other components in the Federate. Another issue is that syntactic errors are discovered much later in the development cycle.

#### 3.2.2 Support for Class Hierarchy

[1] The RTI v0.X series used during the HLA proof-of-concept efforts (expected to end mid-August 1996) was implemented with CORBA servers for each RTI interface. RTI v1.0 will be released with a C++ API and linked in library.

[2] A *Facade* is an object-oriented design pattern which provides a unified higher-level interface combining a set of sub-system interfaces which results in an easier to use interface.

The HLA Declaration Management services allow federates to perform type based filtering by subscribing to data of a particular class type. The service has been defined such that a federate will receive updates from instances of remote object classes the federate subscribed to, as well as updates from instances of derived classes. This behavior is as expected but requires additional support due to the details of how a federate is notified of a remote object instance. When a remote object that meets a federate's subscription criteria is instantiated and updated, the *Discover Object* service [1] is invoked on that federate. This service supplies the class handle (class type id), object instance id, and the initial set of subscribed attribute ids and values. When the discovered object is a derived class of a class the federate is subscribed to, the federate may require additional services to understand the class of data. For example, the federate may know that class AirVehicle exists but not know its subclass, F-16. The federate must be able to determine Is-A relationships[3] based on the RID of the current federation execution.

### 3.2.3 FCS Approach

To facilitate interoperability within a federation the FCS parses the FOM and uses the information as the data store for all RTI and federate data representation and typing information. The FOM Manager class reads the OMT specified FOM tables (Class Structure, Attribute, and Interaction currently) and creates an O-O database (in memory). The resulting database is made available to the application through the query services in the FOM Manager and other supporting class interfaces. The FOM Manager provides additional interfaces to generate the RTI RID.

Upon instantiation of the Federate class, the FCS will create and join the federation, query the RTI for the RTTI for each class/attribute and interaction/parameter name, and publish/subscribe to the FOM data. Due to time constraints, the FCS currently publishes and subscribes to all data specified in the FOM tables. Additional information allowing the FCS to discriminate between the information it should subscribe and publish for each different federate is included in the JPSD FOM; however, the FOM Manager has yet to be extended to act on that information.

### 3.3 Translation Framework

To facilitate the interoperation of legacy systems and to help optimize the mapping of compile-time type of class and attributes to the HLA run-time type identification a small framework of cooperating classes was developed. The right half of the diagram in figure 2 shows the major objects involved in the translation framework. The translation framework consists of an Evaluator class and Object and Interaction Actor classes and is supported by the FOM Manager services defined previously.

The Evaluator object is the global interface where the SimAmbassador (RTI) and Simulation provide data to the FCS. The Evaluator maintains a collection of Actor subclass objects that have been registered by the application developer for translating between the simulation and federation data representation. When a developer defines a specialized Actor class (subclass) a documented process is followed. This process involves defining the class and attribute names the Actor can translate and providing an enumerated index of the class and attribute names to the base Actor class. At instantiation time an Actor, in conjunction with the FOM Manager services, uses this information to build a map[4] to translate between the simulation type identification for the name and the HLA run-time type identification. The arrays of names provide the Actor object with the needed information to automatically support the Actor::pass(ObjectClass) method that the Evaluator uses to decide which Actor can process the reflected data from the RTI. The enumerations are a programmer's aid for implementing the process method which reconstitutes the simulation data structure from the HLA AttributeNameValuePair.

An example of an Actor and subclass definitions:

```
class HOActor
{
public:
    //---------------------------------------------------------------------------
    // Instances automatically register with Evaluator upon
    // instantiation
    //---------------------------------------------------------------------------
    friend HlaEvaluator;

    HOActor();
```

[3] An Is-A relationship is an object-oriented inheritance relationship. For example, if class Dog inherits from class Animal, it is said that an instance of Dog "Is-A" Animal.

[4] The map (SimToRtiMap class) is a direct index lookup and has complexity of O(1), meaning that the lookup time is constant (and small), therefore does not increase with the number of Actor objects, simulation type names, or RID names.

```
    HOActor( const char** classNames, const char** attributeNames );

    ~HOActor();

    virtual int pass( rti_types::ObjectClass id );

    //-------------------------------------------------------------------------
    // Abstract methods - these must be implemented in subclass
    //-------------------------------------------------------------------------
    virtual int process(rti_types::ObjectClass                  classId,
                        rti_types::ObjectId                     sender,
                        rti_types::AttributeNameValuePairSetavList ) = 0;

    virtual int process(rti_types::ObjectClass                  classId,
                        rti_types::ObjectId                     sender,
                        rti_types::AttributeNameValuePairSetavList,
                        rti_types::WallClockTime                theTime )= 0;

    // Protected & private infrastructure stuff ...
};

class SSFHoActor : public HOActor
{
public:
    int process(    rti_types::ObjectClass                  classId,
                    rti_types::ObjectId                     sender,
                    rti_types::AttributeNameValuePairSetavList );

    int process(    rti_types::ObjectClass                  classId,
                    rti_types::ObjectId                     sender,
                    rti_types::AttributeNameValuePairSetavList,
                    rti_types::WallClockTime                theTime );

    int process(    rti_types::ObjectId objectId, SSFEvent *pdu );
};

const char *ssfClasses[] = {
  "ENTITY",
  NULL
};

enum ssfClassEnums
{
   SSFHoActor_TANKS=0,
   SSFHoActor_DISMOUNTED_INFANTRY,
   SSFHoActor_IFVS
};

const char *ssfAttributes[] = {
   "UnitMarking",
   "TimeStamp",
   "EntityType",
   "VehicleId",
   "LocationX",
   "LocationY",
   "LocationZ",
   "VelocityX",
   "VelocityY",
   "VelocityZ",
    NULL
};

enum ssfAttributeEnums {
   SSFHoActor_UnitMarking = 0,
   SSFHoActor_TimeStamp,
   SSFHoActor_EntityType,
   SSFHoActor_VehicleId,
   SSFHoActor_LocationX,
   SSFHoActor_LocationY,
   SSFHoActor_LocationZ,
   SSFHoActor_VelocityX,
   SSFHoActor_VelocityY,
   SSFHoActor_VelocityZ,
   SSFHoActor_MAXVALUE
};

SSFHoActor::SSFHoActor() : HOActor  ( ssfClasses, ssfAttributes )
{
}
```

**Example 1: RAT Entity Actor**

Example 1 defines an *SSFHoActor* class which inherits from the HoActor base class. As seen in the *ssfClasses* array the SSFHoActor class is registered to translate a HLA FOM/RID Entity class and any derived classes of Entity. The *ssfAttributes* array specifies the attributes within the Entity class that SSFHoActor can understand.

A parallel example could be given for HLA Interactions. *Receive Interaction* service [1] invocations are handled by the Evaluator and HIActors.

### 3.4 Measure of Performance (MOP) Services

In support of DMSO's prototyping of the HLA, the FCS was extended with facilities to measure and collect performance statistics. One of the major design goals was to develop an architecture which was as non-intrusive as possible. The RtiAmbassador and SimAmbassador classes were instrumented to send logging information to the MOP Manager which enqueues them for a separate light weight process thread to store. The Logging thread pops events from the queue and writes them to a local disk store (See figure 3).

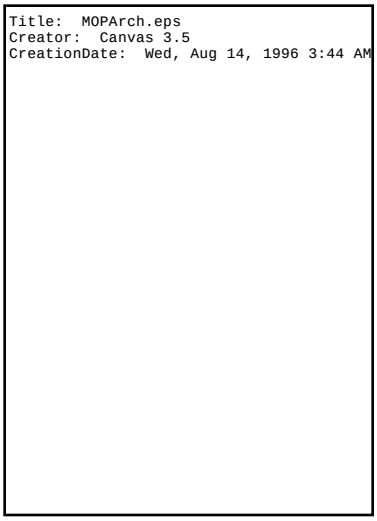


**Figure 3: MOP Architecture**

For more information on the MOP architecture or results of experimentation at the HLA Testbed see [6].

## 4.0 APPLICATIONS

The FCS is currently being used in the JPSD Experiment, HLA Integration & Evaluation Center (IEC) Testbed, and STOW Simulation Support Framework. Applications that have been developed or integrated to work with the FCS and HLA include: CLCGF ModSAF/SIU, DIS Gateway (DGW), Tactical Gateway (TGW), and the RTI Analysis Tool (RAT).

## REFERENCES

[1] Defense Modeling and Simulation Office, **HLA Interface Specification 1.0**, 8/96
[2] Defense Modeling and Simulation Office, **HLA Object Model Template 1.0**, 8/96
[3] Defense Modeling and Simulation Office, **HLA Rules 1.0**, 8/96
[4] Object Management Group, **Common Object Request Broker: Architecture and Specification**, 7/95, p.1-1
[5] E. Gamma et al., "Structural Patterns, **Design Patterns**, 1995, p. 185
[6] J. Olszewski et al., "HLA Testbed Declaration Management Experiments" **15th Workshop on Standards for the Interoperability of Defense Simulations**, Orlando, FL, September 1996, ASD-15-096

## ABOUT THE AUTHORS

**Richard A. Briggs** is a Systems Engineer with Virtual Technology Corporation. He received his B.Sc. in Computer Science from the Pennsylvania State University. He is currently the VTC lead for the RTI Integrated Product Team which is tasked to productize the RTI for the 1.0 release. He was the lead engineer for the JPSD Experiment and the Federation Common Software.

**Gordon J. Miller** is a software engineer with Science Application International Corporation, and is currently working on several HLA related projects. He received his M.S. degree in Physics from George Mason University.